# 4ビット・ミニコンピューターについて (数学とくに整数論,組合せ問題などの超大型計算)

AUTHOR(S):

高橋, 義造

CITATION:

高橋, 義造. 4ビット・ミニコンピューターについて (数学とくに整数論,組合せ問題などの超大型計算). 数理解析研究所講究録 1972, 155: 74-97

ISSUE DATE:

1972-08

URL:

http://hdl.handle.net/2433/106842

数理解析研究所講究録
155巻 1972年 74-97

71

# 4ビット・ミニコンピュータについて

東芝 電子計算機事業部
高橋義造

## 1. はじめに

超大型計算を主題とするこの研究集会に，超小型計算機の話をもち出すのは非常に場違いな感がするが，主催者の方の深いお考えを推量するよしもないので，この発表をお引き受けした次第である。そこで本日の発表が超大型計算に関係があるかどうかは考えないことにして，4ビットミニコンピュータの話をさせて頂くことにする。

コンピュータの大きさをはかるのに，演算速度，主記憶容量，レジスタ数，語長，命令数，入出力機器の性能，チャンネル数，チャンネルのデータ転送速度などの尺度があるが，値段をもってするのがもっとも手っとり早い。コンピュータの値段と性能の間には有名なGroschの法則，即ち

$$\text{Performance} \propto (\text{price})^2$$

があるとされるので値段と性能には一意的な関係がある。

しかしながら，超大型電子計算機の出現，ソフトウェア危機（つまりいかに人員を投入しても，一定限度以上大きなソフトウェアをつくることはできないという警告），および，ミニコンピュータの発達によって図1のように，A（超大型機），C（ミニコンピュータ）のようなGroschの法則の成立しない領域があらわれた。

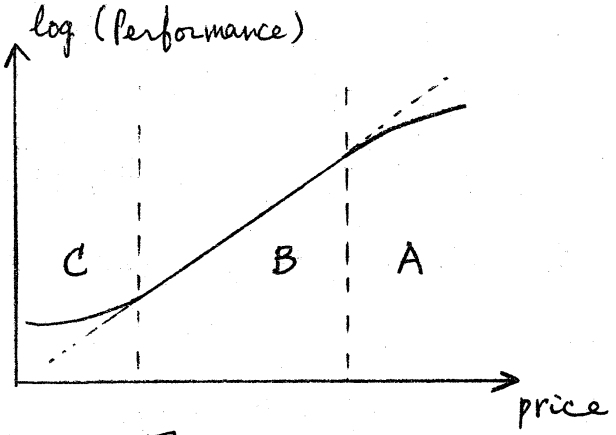



図1

図1のようにコンピュータはそのスケールを小さくしても，性能はさして減少しない。ミニコンピュータでも使いようによっては中，小型機に近い効果を出せるわけである。従ってCの領域をさらに左へ拡げることにより，非常に小さなミニコンピュータでも予想外の力を発揮する可能性がうかがわれる。ミニコンの特徴を一言でいえば，コンピュータを構成する演算制御装置（CPU），主記憶装置（メモリ），周辺装置のうち，CPUとメモリが極端に廉いということである。

よってミニコンの効果を発揮するのは，I/Oよりも CPU をよく使う計算を行なう場合である。

## 2．4ビットマシンの発想

コンピュータの内部における情報の単位は1か0の値をもつビットであるが，処理される情報の単位はビットの集合である語（word）である。語には命令語とデータ語がある。命令語とデータ語の語長は必らずしも等しくはないが，通常は基本命令の語長と整数データの語長は等しく，命令語の中にこれの整数倍のものがあり，浮動小数点数の語長は整数データの2倍などになっていることが多い。

次に主なコンピュータの基本語長を示す。

| | |
|---|---|
| CDC－6600 | 60ビット |
| IBM 7094 | 36ビット |
| IBM S/360 | 32ビット |
| HITAC－5020 | 48ビット |
| TOSBAC－3400 | 24ビット |

また，ミニコンピュータでは次のようなものがある。

| | |
|---|---|
| PDP－8 | 12ビット |
| HITAC－10 | 16ビット |
| TOSBAC－40 | 16ビット |

TOSBAC-10　　8ビット

VARIAN 520/i　　8ビット

当初は8ビットの語長というのは，とうてい使用にたえないとおもわれていたが，8ビットマシンでもFORTRANコンパイラが実現されており，決して16ビットマシンに劣らない性能をもっている。そこでこれに力をえて，さらに，語長が短かくならないかと考えてみたくなる。

語長が短かくなることの利点は，演算回路が簡単になること，デコーダが簡単になること，データパスのゲート数が減少することなどのために，CPUのコストが非常に廉くなること，および回路の簡素化の結果として演算の高速化がはかれることである。

それでは，どこまで語長は短かくできるか？　6ビットあるいは4ビットが考えられる。もっとも1ビットでもよいがもしれないが（Turingマシン），実際には4ビットぐらいが限度であろう。語長を4ビットにすると $2^4=16$ 通りの組合せが可能になり，

1．0～9の10進数が表現できる。

2．プッシュホンのような12ケの入力キーでデータが表現できる。

3．ディジタルプリンタのような簡単な出力装置で

16桁にわたり約16種類の文字を指定できる。

4. load, store など基本命令数16ケ以内で何とか足りる。

のような条件が満たされる。

このような検討により4ビットマシンが可能であることがわかった。そこで本年1月の箱根でのプログラミング・シンポジウムの自由討議で話題にしたところ，東大穂坂教授のミニミニコンピュータ論によって同じような考えのあることを知った。たまたまこの頃米国の半導体メーカINTEL社よりLSIによる4ビットコンピュータMCS-4（Micro-Computerとよばれる）が発表され，カタログが配布されはじめた。これは次章にのべるように4ビットのデータ語を取り扱かうもので，Cash Registerや電卓などの分野に100ドルコンピュータとして売り込まれている。また，前記の自由討議の席で学習院大学米田教授からSonyのMicro-Computer SOBAX ICC-2700も4ビットマシンであることを教えていただいた。SOBAXのデータ語は，10進15桁＋符号（64ビット）語長であるが，命令語が4ビットである。これにたしINTELのMCS-4の命令語の語長は8ビットである。MCS-4は発表されたばかりでその応用分野はまだ開拓されておらず，効果のほどは未知

数であるが，SOBAXは高級電卓として知られており，愛好者も少なくないようである。

| 機種 / 語長 | INTEL MCS-4 | SOBAX ICC-2700 |
|---|---|---|
| データ語 | 4ビット | 64ビット |
| 命令語 | 8ビット | 4ビット |

それではデータ語と命令語の語長がともに，4ビットの本当の4ビットマシンは可能であろうか。一つの試案を示すことにするが，その前にMCS-4とSOBAXの紹介を行なうこととする。

## 3. INTELの4ビットマシン MCS-4

MCS-4は次の4つの基本的なLSIチップより構成される。

1. CPU　4004
2. ROM (read only memory)　4001
3. RAM (random access memory)　4002
4. SR (shift register)　4003

1つのCPUにつき16ヶのROMと16ヶのRAMを接続することができる。SRは入出力を拡張するために必要に応

じて使用される。上記の4つの要素は次の図のように接続される。

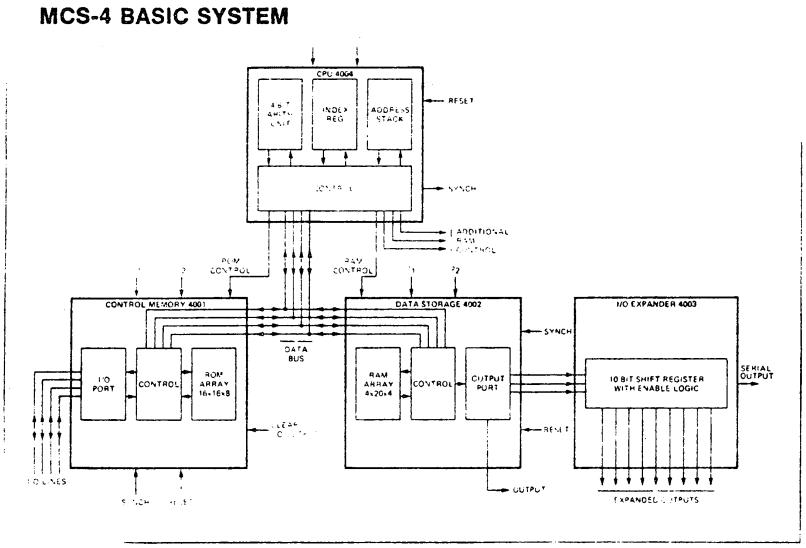



図 2

3.1 ROM 4001

MCS-4のプログラムはROMに記憶される。4001はmask programmableなROMで1ヶで256×8ビットの情報を記憶させることができる。CPUとつながれている4本のdata bus lineにより，ROMアドレスとデータがやりとりされる。1つのinstruction cycleは10.8μsで8ヶのclock cycleより構成されているが，最初の3サイクルの間にCPUよりROMに対して12ビットの（4ビットずつ3回にわけて）ROMのアドレスが送られる。ROMの1語は8ビットで，この中に書き込まれている命令語は次の2サイクルで（4ビットずつ2回にわけて）ROMよりCPUにおくられる。残りの3サイクルでこの命令が解釈されて実行される。ROMにはまた，I/O portがあり，CPUの指示によりdata bus lineを通してCPUから送られる4ビットの情報を出力したり逆に入力データをCPUにおくる仕事を行なう。1つのCPUに最大16ヶのROMがつながれるので4Kバイト（1バイト＝8ビット）の記憶容量がプログラムの記憶に使用される。

3.2 RAM 4002

RAMはデータの記憶，およびデータの出力に使用される。これはそれぞれ4ビットずつの16ヶのmemory character

と4ヶのstatus characterよりなる合計 20×4ビットのレジスタ4ヶより構成されたRAMで1つのCPUには16ケのRAMが接続されるので，全部で1280×4ビットの記憶容量となる。

CPUがSRC（send address to ROM or RAM）命令を実行すると8ビットのRAM addressが2回にわけておくられて，1つのmemory characterが指定される。なお，SRC命令に先立ってDCL（designate Common and line）命令によって16ケの4002のうち4ケのRAM bankを選択しておかねばならない。つぎにWRM，ROMなどの命令の実行によりRAMとCPU間のデータの転送が行なわれる。4002にはまた，4本の出力ラインをもつoutput portがあり，WMP命令によりCPUのアキュムレータの内容を出力することができる。

3.3 SR 4003

4003は10ビットのシフトレジスタで4本しかないROMあるいはRAMの入力ラインを10本まで拡張するために使用される。これに10ビットのシフトレジスタで serial input, serial output, parallel outputをもつ。

3.4 CPU 4004

CPUには4ビットのadder，16ケの4ビットのindex

register, 4ケの12ビットのprogram counter stack, 8ビットのinstruction registerなどがある。1インストラクションサイクルは10.8μsであり，8桁の10進数の加算は850μsで行われる。4004には44ケの命令があり，そのうち5ケは16ビットの語長，他はすべて8ビットの語長である。その命令はつぎの通りである。

[Those instructions preceded by an asterisk (*) are 2 word instructions that occupy 2 successive locations in ROM

MACHINE INSTRUCTIONS

| MNEMONIC | OPR $D_3 D_2 D_1 D_0$ | OPA $D_3 D_2 D_1 D_0$ | DESCRIPTION OF OPERATION |
|---|---|---|---|
| NOP | 0 0 0 0 | 0 0 0 0 | No operation. |
| *JCN | 0 0 0 1 | $C_1 C_2 C_3 C_4$ | Jump to ROM address $A_2 A_2 A_2$, $A_1 A_1 A_1 A_1$ (within the same ROM that contains this JCN instruction) if condition $C_1 C_2 C_3 C_4$ (1) is true, otherwise skip (go to the next instruction in sequence). |
| | $A_2 A_2 A_2 A_2$ | $A_1 A_1 A_1 A_1$ | |
| *FIM | 0 0 1 0 | R R R 0 | Fetch immediate (direct) from ROM Data $D_2, D_1$ to index register pair location RRR. (2) |
| | $D_2 D_2 D_2 D_2$ | $D_1 D_1 D_1 D_1$ | |
| SRC | 0 0 1 0 | R R R 1 | Send the address (contents of index register pair RRR) to ROM and RAM at $X_2$ and $X_3$ time in the Instruction Cycle. |
| FIN | 0 0 1 1 | R R R 0 | Fetch indirect from ROM. Send contents of index register pair location 0 out as an address. Data fetched is placed into register pair location RRR at $A_1$ and $A_2$ time in the Instruction Cycle. |
| JIN | 0 0 1 1 | R R R 1 | Jump indirect. Send contents of register pair RRR out as an address at $A_1$ and $A_2$ time in the Instruction Cycle. |
| *JUN | 0 1 0 0 | $A_3 A_3 A_3 A_3$ | Jump unconditional to ROM address $A_3, A_2, A_1$. |
| | $A_2 A_2 A_2 A_2$ | $A_1 A_1 A_1 A_1$ | |
| *JMS | 0 1 0 1 | $A_3 A_3 A_3 A_3$ | Jump to subroutine ROM address $A_3, A_2, A_1$, save old address. (Up 1 level in stack.) |
| | $A_2 A_2 A_2 A_2$ | $A_1 A_1 A_1 A_1$ | |
| INC | 0 1 1 0 | R R R R | Increment contents of register RRRR. (3) |
| *ISZ | 0 1 1 1 | R R R R | Increment contents of register RRRR. Go to ROM address $A_2, A_1$ (within the same ROM that contains this ISZ instruction) if result ≠ 0, otherwise skip (go to the next instruction in sequence). |
| | $A_2 A_2 A_2 A_2$ | $A_1 A_1 A_1 A_1$ | |
| ADD | 1 0 0 0 | R R R R | Add contents of register RRRR to accumulator with carry. |
| SUB | 1 0 0 1 | R R R R | Subtract contents of register RRRR to accumulator with borrow. |
| LD | 1 0 1 0 | R R R R | Load contents of register RRRR to accumulator. |
| XCH | 1 0 1 1 | R R R R | Exchange contents of index register RRRR and accumulator. |
| BBL | 1 1 0 0 | D D D D | Branch back (down 1 level in stack) and load data DDDD to accumulator. |
| LDM | 1 1 0 1 | D D D D | Load data DDDD to accumulator. |

**Table V - Basic CPU Instruction Set**

## INPUT/OUTPUT AND RAM INSTRUCTIONS

(The RAM's and ROM's operated on in the I/O and RAM instructions have been previously selected by the last SRC instruction executed.)

| MNEMONIC | OPR $D_3 D_2 D_1 D_0$ | OPA $D_3 D_2 D_1 D_0$ | DESCRIPTION OF OPERATION |
|---|---|---|---|
| WRM | 1 1 1 0 | 0 0 0 0 | Write the contents of the accumulator into the previously selected RAM main memory character. |
| WMP | 1 1 1 0 | 0 0 0 1 | Write the contents of the accumulator into the previously selected RAM output port. (Output Lines) |
| WRR | 1 1 1 0 | 0 0 1 0 | Write the contents of the accumulator into the previously selected ROM output port. (I/O Lines) |
| WRϕ[4] | 1 1 1 0 | 0 1 0 0 | Write the contents of the accumulator into the previously selected RAM status character 0. |
| WR1[4] | 1 1 1 0 | 0 1 0 1 | Write the contents of the accumulator into the previously selected RAM status character 1. |
| WR2[4] | 1 1 1 0 | 0 1 1 0 | Write the contents of the accumulator into the previously selected RAM status character 2. |
| WR3[4] | 1 1 1 0 | 0 1 1 1 | Write the contents of the accumulator into the previously selected RAM status character 3. |
| SBM | 1 1 1 0 | 1 0 0 0 | Subtract the previously selected RAM main memory character from accumulator with borrow. |
| RDM | 1 1 1 0 | 1 0 0 1 | Read the previously selected RAM main memory character into the accumulator. |
| RDR | 1 1 1 0 | 1 0 1 0 | Read the contents of the previously selected ROM input port into the accumulator. (I/O Lines) |
| ADM | 1 1 1 0 | 1 0 1 1 | Add the previously selected RAM main memory character to accumulator with carry. |
| RDϕ[4] | 1 1 1 0 | 1 1 0 0 | Read the previously selected RAM status character 0 into accumulator. |
| RD1[4] | 1 1 1 0 | 1 1 0 1 | Read the previously selected RAM status character 1 into accumulator. |
| RD2[4] | 1 1 1 0 | 1 1 1 0 | Read the previously selected RAM status character 2 into accumulator. |
| RD3[4] | 1 1 1 0 | 1 1 1 1 | Read the previously selected RAM status character 3 into accumulator. |

## ACCUMULATOR GROUP INSTRUCTIONS

| | | | |
|---|---|---|---|
| CLB | 1 1 1 1 | 0 0 0 0 | Clear both. (Accumulator and carry) |
| CLC | 1 1 1 1 | 0 0 0 1 | Clear carry. |
| IAC | 1 1 1 1 | 0 0 1 0 | Increment accumulator. |
| CMC | 1 1 1 1 | 0 0 1 1 | Complement carry. |
| CMA | 1 1 1 1 | 0 1 0 0 | Complement accumulator. |
| RAL | 1 1 1 1 | 0 1 0 1 | Rotate left. (Accumulator and carry) |
| RAR | 1 1 1 1 | 0 1 1 0 | Rotate right. (Accumulator and carry) |
| TCC | 1 1 1 1 | 0 1 1 1 | Transmit carry to accumulator and clear carry. |
| DAC | 1 1 1 1 | 1 0 0 0 | Decrement accumulator. |
| TCS | 1 1 1 1 | 1 0 0 1 | Transfer carry subtract and clear carry. |
| STC | 1 1 1 1 | 1 0 1 0 | Set carry. |
| DAA | 1 1 1 1 | 1 0 1 1 | Decimal adjust accumulator. |
| KBP | 1 1 1 1 | 1 1 0 0 | Keyboard process. Converts the contents of the accumulator from a one out of four code to a binary code. |
| DCL | 1 1 1 1 | 1 1 0 1 | Designate command line. (See note 1 on page 3.) |

NOTES: [1]The condition code is assigned as follows:

$C_1 = 1$ Invert jump condition — $C_2 = 1$ Jump if accumulator is zero — $C_4 = 1$ Jump if test signal is a 0

$C_1 = 0$ Not invert jump condition — $C_3 = 1$ Jump if carry/link is a 1

[2]RRR is the address of 1 of 8 index register pairs in the CPU.

[3]RRRR is the address of 1 of 16 index registers in the CPU.

[4]Each RAM chip has 4 registers, each with twenty 4 bit characters subdivided into 16 main memory characters and 4 status c Chip number, RAM register and main memory character are addressed by an SRC instruction. For the selected chip and register, status character locations are selected by the instruction code (OPA).

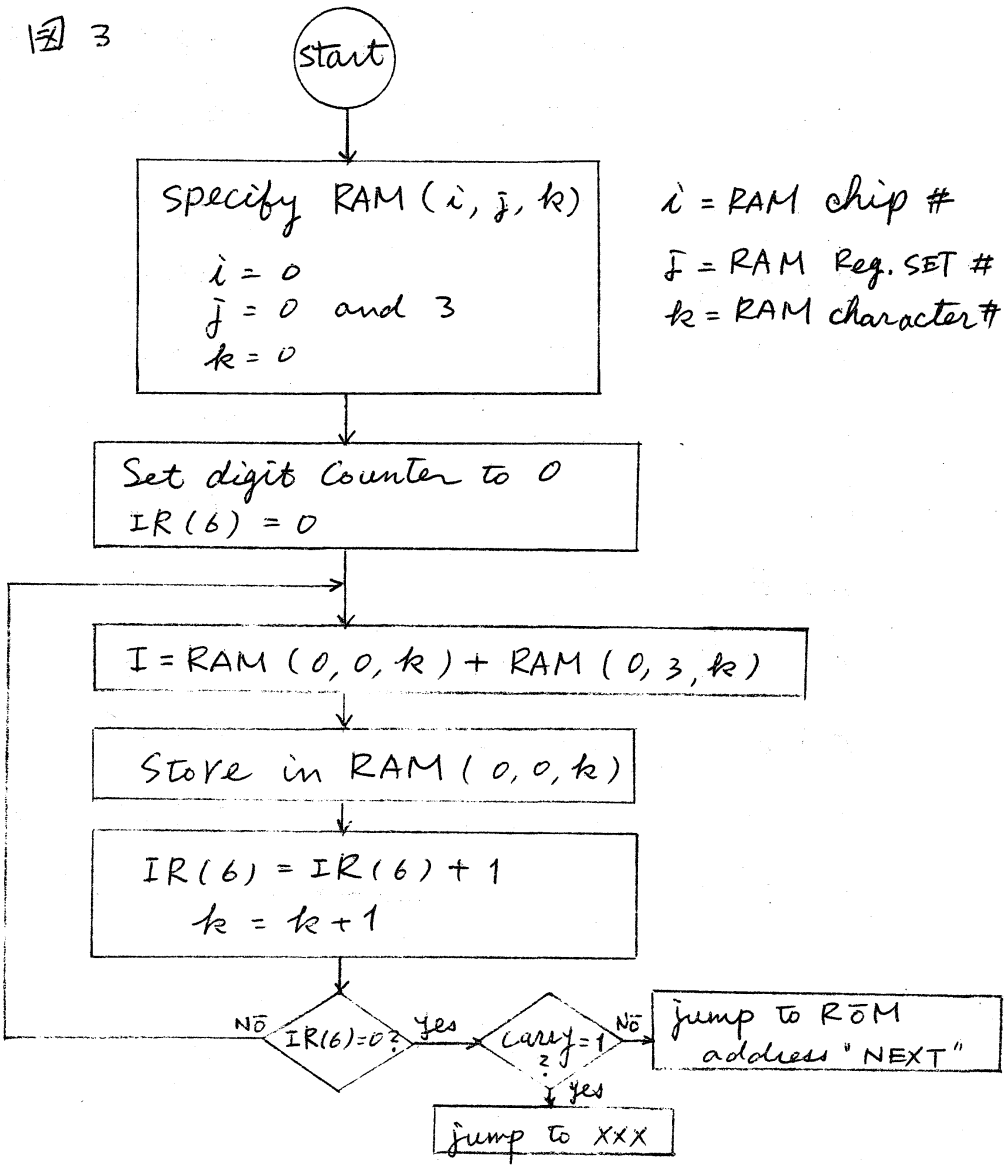
Table V - Basic CPU Instruction Set (Continued)

3.5 プログラム例

RAM 0 の Register 0 と 3 に入っている 16 桁の 10 進数を加算して Register 0 に入れるプログラムを書いてみる。

RAM register の memory character 0 には最小桁が 15 には最大桁が入っているものとする。

図 3



| location | instruction word | | mnemonic | |
|---|---|---|---|---|
| 000 | 2000 | | FIM | 0, 0 |
| 002 | 2430 | | FIM | 4, '30' |
| 004 | D0 | | LDM | 0 |
| 005 | B6 | | XCH | 6 |
| 006 | F1 | | CLC | |
| 007 | 25 | AD1 | SRC | 4 |
| 008 | E9 | | RDM | |
| 009 | 21 | | SRC | 0 |
| 00A | EB | | ADM | |
| 00B | FB | | DAA | |
| 00C | E0 | | WRM | |
| 00D | 61 | | INC | 1 |
| 00E | 65 | | INC | 5 |
| 010 | 76 07 | | ISZ | 6, AD1 |
| 012 | 12 15 | | JCN | 2, *** |
| 014 | 401C | | JUN | NEXT |
| 015 | | *** | -------- | |

4. SOBAX ICC－2700

SOBAX ICC－2700は高級電卓の1つで，MANUALモードではキーを押すことにより演算が直接行われ，AUTOモードではプログラムメモリに記憶された最大253ステップのプログラムが自動的に実行される。データの記憶には12ケのデータ・メモリが用意されており，10進15桁＋符号のデータが記憶される。データキーより入力されたデータあるいはデータ・メモリより読み出されたデータはレジスタ1に入るとともに数字表示される。このデータについての演算キーがおされるか，あるいはプログラムメモリから演算コードが読み出されるとデータはレジスタ2におくられる。そして演算数がキーまたはデータメモリより入力されレジスタ1に入る。そして次に演算キーがおされると，先におされた演算キーに対応して，レジスタ2のデータに対しレジスタ1のデータが演算され，レジスタ1に入り表示される。また，同時に演算数であるレジスタ1にあった数はレジスタ2に移される。15桁の10進数の加減算は5msで行なわれる。

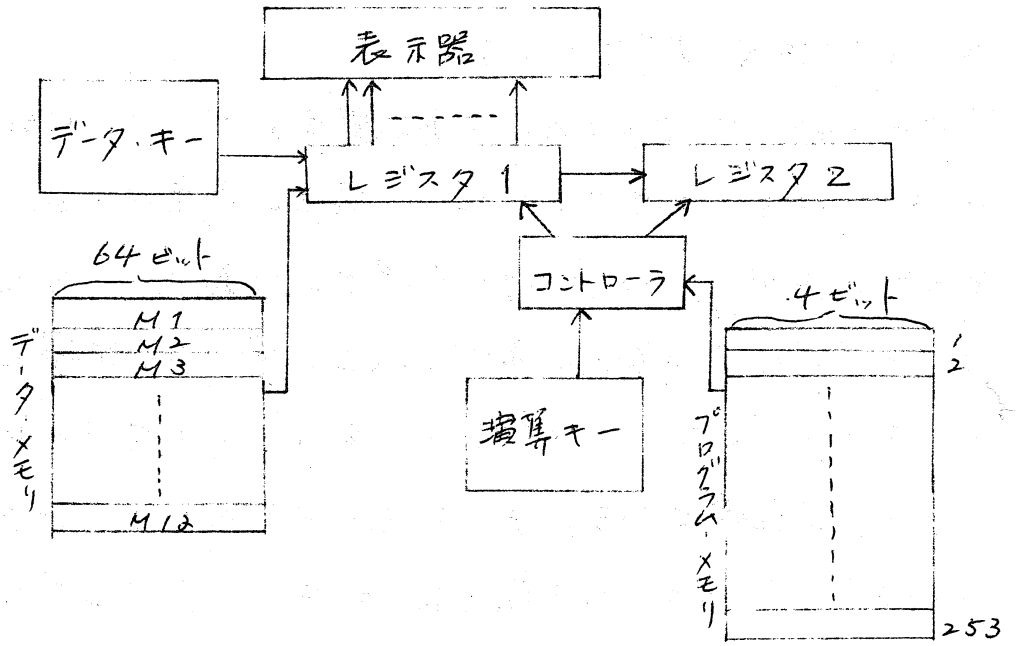



図 4

4.1 レジスタとメモリの語長

| | |
|---|---|
| レジスタ1, レジスタ2 | 10進15桁+符号 |
| データ・メモリ M1～M12 | 10進15桁+符号 |
| プログラム・メモリ | 4ビット |

4.2 命令語

MANUALモードでは演算命令は演算キーを押すことによって発生される。この演算キーの1つ1つが命令語に対応する。

| No. | 演算キー | 演算 |
|---|---|---|
| 1 | + | add |
| 2 | − | Subtract |

| NO. | 演算キー | 演　算 |
|---|---|---|
| 3 | × | multiply |
| 4 | ÷ | devide |
| 5 | = | 演算の表示 |
| 6 | √ | 平方根 |
| 7 | Min | (register 1)+(データメモリ)→データメモリ |
| 8 | Mout | (データメモリ)→register 1 |
| 9 | Mc | 0 → データメモリ |
| 10 | M | データメモリのアドレス指定 |
| 11 | R | (register 1), (register 2)の交換 |
| 12 | () | 数値モードと記号モードの切り換え |
| 13 | CHG, SIGN | −(register 1)→register 1 |
| 14 | S | stop, キーボードよりレジスタ1にデータを入れることができる。 |
| 15 | J | jump |
| 16 | End | end of program |

これらの演算命令は16ヶあり，4ビットで表現される。プログラム中に数字データが入るが，これは( )と( )の間にかこまれた部分，およびM，Jの次の文字として区別される。

4.3 データ語

先にものべたようにSOBAX LC−2700で演算の対象と

なるデータ語は10進数15桁+符号であるが，1つの桁は14種類のデータ・キー，即ち，

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

•, Change sign, ←, →

で表わされる4ビットデータである。

4.4 プログラム例

n個のデータ $x_1, x_2 \cdots\cdots, x_n$（nは任意）の平均値をもとめるプログラムをかく。計算機が停止する度に，それまでの平均値が表示され次のデータ $x_i$ をデータ・キーから入力し，演算キーSを押してスタートするようなプログラムをつくる。

| プログラム | | | | 説明 |
|---|---|---|---|---|
| S | | | | |
| M1 | MC | Min | | input $x_1$ |
| M2 | MC | | | $0 \rightarrow M_2$ |
| ( ) 1 ( ) M3 | | Mc | Min | $1 \rightarrow M_3$ |
| MJ1 | | | | jump point 1 |
| M3 | Mout | | | (M3) ⟶ reg. 1 |
| M2 | Min | | | (reg. 1) + (M2) → (M2) |
| Mout | ÷ | M1 | Mout | |
| R | = | S | | (M1) ÷ (M2) input $x_i$ |
| M1 | Min | | | (reg. 1) + (M1) → M1 |
| − | R | = | J1 | unconditional jump to 1 |
| END | | | | |

# 5. 4ビットマシンの実現

前の2章で2つの実在する4ビットマシンの解説をしたがいずれもデータ語又は命令語のいずれかが4ビットではなかった。そこで命令語もデータ語のいずれもが4ビットであるような4ビットマシンが可能かどうかを考えることとする。

## 5.1 データ語

データ語の語長は4ビットとする。4ビットのデータは16進表示により

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

A, B, C, D, E, F

のコードで表わされる。

10進数のデータは次のようにあらわされる。

| コード | 10進データ |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |
| 5 | 5 |
| 6 | 6 |
| 7 | 7 |
| 8 | 8 |
| 9 | 9 |
| A | delimiter for positive number |
| B | delimiter for negative number |
| C | decimal point |

例

123 ⟶ 123A

−56 ⟶ 56B

123.45 ⟶ 123C45A

−3.14 ⟶ 3C14B

5.2 メモリー

プログラムおよびデータを記憶させるためのメモリに16ケのstackを用いる。番号を一度指定すると，次に指定があるまで同じstackが指定されたこととなる。readされたデータは必らずstackにつまれ，また，stackから読み出されたデータはアキュムレータに入るか，あるいはインストラクションレジスタに入る。stackからデータが読まれると，そのstackに最後に入れられたデータが取り出され，stackからは消される。stackにデータがストアされるとこれは一番上におかれる。一組の表示器が指定のスタックに接続されるとそのstackの内容が下を左にして表示される。

5.3 レジスタ

この4ビットマシンでは演算はすべて10進演算で4ビットのアキュムレータで行なわれる。演算されるデータは16ケのstack（S0～S15）より取り出され，演算の結果はアキュムレータにのこる。レジスタとしてはアキュムレータの外

にMQレジスタ（4ビット），キャリーレジスタ（1ビット），インストラクションレジスタ，スタックアドレスレジスタがある。stackの長さは16語とする。

命令語はstackから読みだされてIRレジスタに入り，デコードされて実行される。

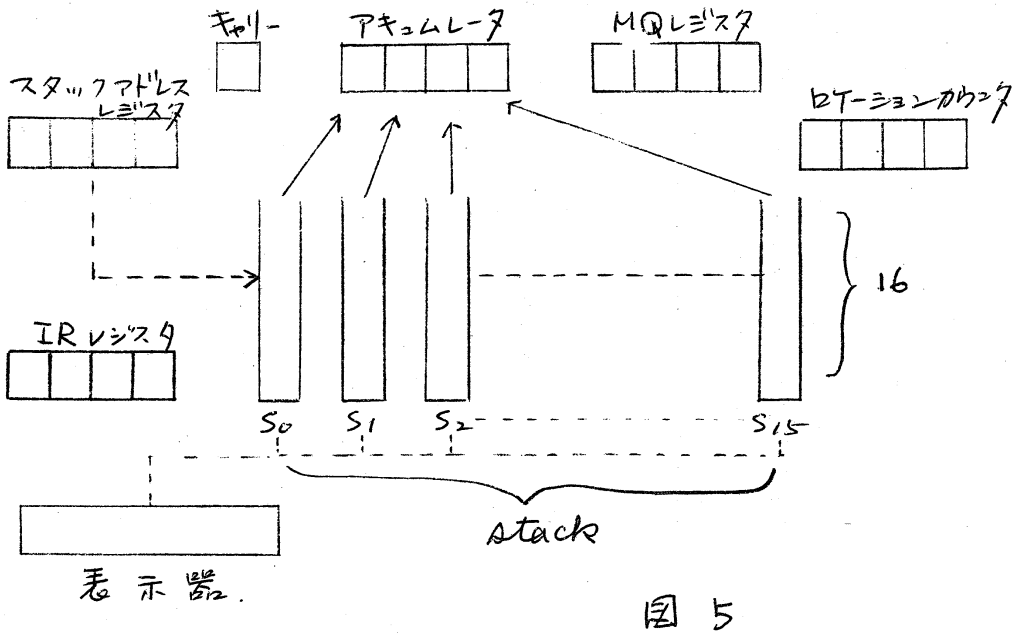



図5

5.4 入出力装置

入出力装置としては下図のような16キーを使用する。

| | | | |
|---|---|---|---|
| 0 | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 |
| 8 | 9 | A | B |
| C | D | E | F |

図6

出力装置

1. stackの内容を表示する16桁以上の数字表示装置があり，命令によって16ヶのstackの任意のものに接続する。stackの一番下に入っているデータが一番左に表示されるようにしておく。

| コード | 表示文字 |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |
| 5 | 5 |
| 6 | 6 |
| 7 | 7 |
| 8 | 8 |
| 9 | 9 |
| A | + |
| B | − |
| C | . |
| D | blank |

2. 表示装置に表示されたデータはディジタルプリンタにプリントされるようにする。ディジタルプリンタを制御する命令はとくにおかない。

5.5 命令語

この4ビットマシンの命令語は4ビットより構成される。特定の命令につづく語はOperandで，アドレス指定等に使われる数値データとみなされる。

命令の種類と命令コードは次の通りである。

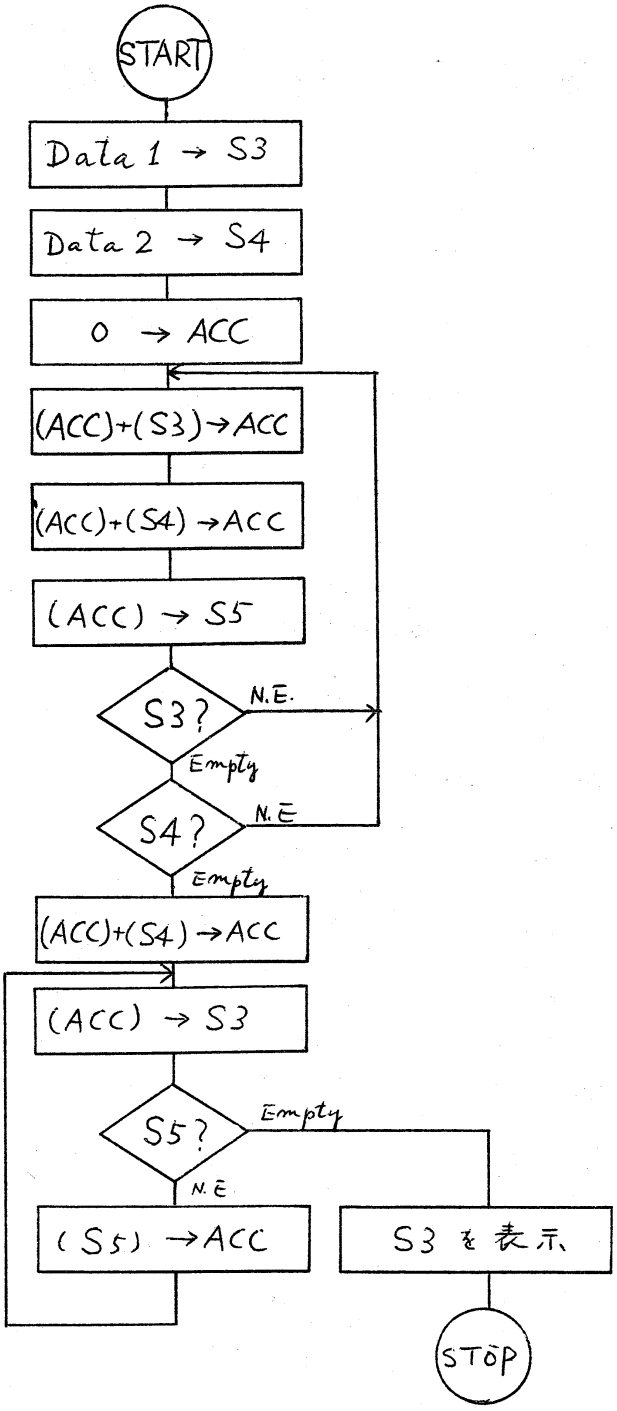
| mnemonic | 命令コード | 機能 |
|---|---|---|
| STOP | 0 | 停止 |
| ADR | 1 | stackの指定，この命令につづくコードでstackを指定する。 |
| STACK | 2 | アキュムレータの内容をstackにつむ。アキュムレータは0になる。 |
| ADD | 3 | スタックの一番上のデータをとってキャリーとともにアキュムレータに加える。 |
| SUB | 4 | スタックの一番上のデータをとって，キャリーとともにアキュムレータより引く。 |
| MULT | 5 | スタックの一番上のデータをとって，アキュムレータの内容にかけ結果をMQレジスタとアキュムレータにおく。 |
| DIV | 6 | スタックの一番上のデータをとってMQレジスタとアキュムレータにある数を割り，商をMQに余りをアキュムレータにおく。 |
| RIGHT | 7 | (MQ) → ACC , (Carry) → MQ |
| LEFT | 8 | (ACC) → MQ , 0 → ACC |
| EMPTY | 9 | stackが空であれば，この命令の次のコードの数だけ命令をスキップする。 |
| JUMP | A | この命令の次のコードで指定されたスタックの，そのスタックの最初のコードの数だけ先の命令にコントロールを移す。 |
| SKIPH | B | (Acc) ≧ 0 のとき，この命令の次のコードの数だけ命令をスキップする。 |
| RESET | C | キャリーをリセットする。 |
| NOP | D | no operation |
| READ | E | キーよりデータを読み，AまたはBがくるまでstackにつみつづける。 |
| SHOW | F | stackに表示器を接続する。 |

プログラムはstack 0から実行される。1つのstackの命令語が処理され，終ると次のstackの処理がはじまる。各stackの最初のデータは必らずスキップされる。このデータはJUMP命令でこのstackが指定されたときに，そのstackにつまれている命令語のうちの何番目から実行に入るかを示す数が入っている。

5.6 プログラム例

キーより2つの16桁以内の10進数を入力し2つの数の和を表示する。但し10進数の終りにはAなるキーを打つものとする。

| LOC | S0 | S1 | S2 |
|---|---|---|---|
| 0 | 8 | 15 | 0 |
| 1 | ADR | STACK | 3 |
| 2 | 3 | ADR | STACK |
| 3 | READ | 3 | ADR |
| 4 | ADR | EMPTY | 5 |
| 5 | 4 | 2 | EMPTY |
| 6 | READ | JUMP | 3 |
| 7 | LEFT | 0 | ADD |
| 8 | ADR | ADR | JUMP |
| 9 | 3 | 4 | 1 |
| A | ADD | EMPTY | ADR |
| B | ADR | 2 | 3 |
| C | 4 | JUMP | STACK |
| D | ADD | 0 | STOP |
| E | ADR | ADD | END |
| F | 5 | ADR | |


START
Data 1 → S3
Data 2 → S4
0 → ACC
(ACC)+(S3)→ACC
(ACC)+(S4)→ACC
(ACC) → S5
S3?
N.E.
Empty
S4?
N.E
Empty
(ACC)+(S4)→ACC
(ACC) → S3
S5?
Empty
N.E
(S5) →ACC
S3 を表示
STOP